

Avoiding The Man on the Wire: Improving Tor's Security with Trust-Aware Path Selection

Aaron Johnson*, Rob Jansen*, Aaron D. Jaggard*, Joan Feigenbaum[†] and Paul Syverson*

*U.S. Naval Research Laboratory, {aaron.m.johnson, rob.g.jansen, aaron.jaggard, paul.syverson}@nrl.navy.mil

[†]Yale University, joan.feigenbaum@yale.edu

Abstract—Tor users are vulnerable to deanonymization by an adversary that can observe some Tor relays or some parts of the network. We propose that users use trust to choose paths through Tor that are less likely to be observed. We present a system to model this trust as probability distributions on the location of the user's adversaries. We propose the Trust-Aware Path Selection algorithm for Tor that helps users avoid traffic-analysis attacks while still choosing paths that could have been selected by many other users. We evaluate this algorithm in two settings using a high-level map of Internet routing: (i) users try to avoid a single global adversary that has an independent chance to control each Autonomous System organization, Internet Exchange Point organization, and Tor relay family, and (ii) users try to avoid deanonymization by any single country. We also examine the performance of trust-aware path selection using the Shadow Tor network simulator.

I. INTRODUCTION

Tor is the most popular tool for low-latency anonymous communication, with over one million daily users. In order to use Tor to communicate with others, clients choose a three-hop path from the set of over 5000 relays volunteering bandwidth to the network. In order to balance load among the relays, and in particular to optimize the allocation of Tor's limited bandwidth, the default path selection algorithm is *bandwidth-weighted* so that a client will select a relay with a probability equivalent to the ratio of that relay's available bandwidth to the total network bandwidth capacity. Clients communicate with arbitrary Internet hosts via a cryptographic *circuit* with a layer of encryption for each of the three relays on its path, which are termed the *entry guard*, *middle*, and *exit*, according to their position on the path. Because this circuit is built using a telescoping process, it provides unlinkability against a passive, non-global adversary that cannot observe both ends of the circuit.

Unfortunately for many Tor users, a global adversary that can observe both ends has become a very real and significant threat. An adversary in such a position can perform a "first-last" traffic-correlation attack for any of the circuit's *streams* (i.e., TCP connections to destinations multiplexed over circuits) by using similarities in the volume and timing of the traffic at both ends to match them with each other, thereby deanonymizing the user. These techniques are efficient and effective [1]. In order to carry out traffic-correlation attacks, an adversary must be in a position to (i) observe Tor traffic on the Internet path between a target client and its chosen entry guard or control than entry guard, and (ii) observe the Internet path between the selected exit relay and the destination or control the exit or destination. Due to Tor's volunteer relay model and its bandwidth-weighted path selection algorithm, an adversary may get in a position to observe a large amount

of traffic simply by running a fast relay, and can otherwise observe traffic by controlling or coercing entities on the paths to and from relays including Internet Service Providers (ISPs), Autonomous systems (ASes), and Internet Exchange Points (IXPs).

Previous approaches to improving resilience against traffic observation and correlation attacks have been limited in nature and only consider specific threats. One main approach focuses on defeating an adversary that observes a single AS or IXP [2]–[6], and suggests AS/IXP *path independence*, that is, creating Tor circuits such that the set of ASes and IXPs that appear on the Internet path between the client and guard is independent of the set of ASes and IXPs between the exit and destination. However, this solution ignores the critical effects of path selection *over time*, resulting in two major flaws: (i) it leaks information about client location with each additional path, and (ii) it either requires that users expose themselves to an increasing number of entry guards or denies them access to some Internet destinations. The other main approach focuses on an adversary that can observe some Tor relays [7], [8] and suggests that, for the most sensitive circuit positions, users choose a small number of relays from among those the user trusts the most. However, this approach leaks information to an adversary that can eventually identify the trusted relays (e.g., via a congestion attack [9], [10]) and uses a restrictive trust model. No existing solution to traffic correlation attacks provides protection from the variety of attacker resources and tactics that recent research and experience suggests is realistic [11].

In contrast, this paper develops defenses against traffic correlation that are based on a completely general probabilistic model of network adversaries. Using this model we can consider adversaries with diverse resources, such as those that observe network traffic at any combination of network providers, exchange points, physical cables (undersea or elsewhere), and Tor relays. The model allows us to incorporate uncertainty about and randomness in an adversaries' actions. It also enables us to express common factors underlying the adversaries' ability to compromise different network locations, such as shared legal jurisdiction or corporate ownership. A user expresses a *trust belief* about her adversaries by specifying a probability distribution over their presence at a set of network locations, and she turns this belief into a *trust policy* by including a weight for each adversary indicating her relative level of concern about that adversary.

Using these trust policies, we design Trust-Aware Path Selection (TAPS), a novel trust-aware path selection algorithm that uses trust in network nodes and elements to inform a user's decision about how to select relays for its Tor path. Using

TAPS, clients select paths so as to minimize the probability that an adversary is in a position to observe both ends of their Tor circuit while at the same time ensuring that their path selection behavior does not harm their security by making them stand out from other clients. In addition to defending against a much broader class of adversaries, TAPS addresses the other deficiencies of prior proposals. In particular, it takes into account the potential ability of an adversary to monitor Tor for extended periods of time, to influence the destinations visited by the client, to link together different Tor connections made by the same client, and to eventually identify relays used repeatedly by a client.

In order to facilitate the adoption of TAPS, we describe both a long-term and a short-term deployment strategy. In the long-term strategy, all Tor users participate and use the *TrustAll* version of TAPS to replace Tor’s existing bandwidth-weighted algorithm. In the short-term strategy, TAPS provides the option for security-conscious users to use trust to defend against traffic-correlation attacks while most users continue to use “vanilla” Tor (*i.e.*, bandwidth-weighted path selection). We design the *TrustOne* TAPS version for this case, in which users must both avoid traffic correlation and choose paths that blend in with the vanilla Tor users.

We evaluate the security of TAPS via simulation with modified versions of the Tor Path Simulator (TorPS) [11]. This evaluation is done with respect to two plausible and illustrative trust policies: (i) The *Man* policy, in which a single adversary has an independent probability of compromising each AS organization (*i.e.*, group of ASes run by the same entity), IXP organization, and self-declared *relay family* in Tor, and (ii) the *Countries* policy, in which each country is considered a potential adversary and observes all ASes, IXPs, and relays located inside of it. Our analysis of The *Man* policy for a popular Tor client location shows a reduction in the probability of a successful first-last attack from about 0.7 to about 0.4 in *TrustAll* with typical web-browsing activity over the first week in December 2013, and from about 0.68 to as little as 0.1 in *TrustOne* with repeated connections over the same week to a single IRC server popular with Tor developers. Our analysis of the *Countries* policy over that week shows a reduction in the median number of countries that “unnecessarily” compromise a stream (*i.e.*, compromise a stream when they don’t contain both the client and destination) from 5 to 2 in *TrustAll* with typical user behavior.

Our algorithms are designed not only to improve security, but also to enable allow security to be traded off for performance. They achieve this by allowing clients to configure the fraction of bandwidth weight that their candidate set of guards and exits should exceed before a bandwidth-weighted choice is made. This mechanism results in a client selecting from among the most secure relays while still making use of much of the network and doing so a bandwidth-weighted manner. We explore these trade-offs using the Shadow simulator [12], [13] and find that there exist parameters that result in only a slight decrease in performance, and only for less than 5 percent of users.

II. TRUST MODEL

A. Trust Policies

We use the definition of network trust given by Jaggard *et al.* [14]. A trust belief is a probability distribution that indicates how likely adversaries are to observe traffic at

different network locations. A location is considered to be more trusted the less likely it is that adversaries are observing it. Although the belief distribution may be expressed as the adversaries’ success in compromising any number of relevant factors, such as relay software or physical location, ultimately it must describe the probability that the adversaries observe traffic on the *virtual links* into and out of the Tor network. A virtual link is an unordered pair of network hosts, and the *entry virtual links* into Tor consist of client-guard pairs while the *exit virtual links* out of Tor consist of destination-exit pairs. An adversary is considered to observe a virtual link if it can observe traffic in at least one direction between the two hosts. Although the representation of an arbitrary distribution over all virtual links can be very large, Jaggard *et al.* [14] describe how distributions of likely Tor adversaries can be represented efficiently by aggregating host locations (*e.g.*, at the AS level) and by identifying a small set of relevant compromise factors and indicating their dependencies in a Bayesian network.

To indicate how to trade off vulnerability to different adversaries, each user adopts a trust policy that pairs her trust belief with a weight for each adversary. Each weight is a number in $[0, 1]$ that indicates the relative level of concern that the user has for the associated adversary. In this work, we assume that distinct adversaries do not collude. If a user were worried about two adversaries colluding, she could combine her beliefs about them into those for a single adversary.

Trust policies are quite general and can easily be used with many kinds of beliefs and sources of trust information. For example, previous work that considered each relay to have an independent and individual probability of compromise [7], [8] can be represented as a trust policy by including a single adversary and allowing him to compromise each relay independently with its given probability. As another example, previous work that considered as a potential threat each AS and IXP [2], [3], [5], [6] can be represented as a trust policy by including each AS and IXP as an adversary with equal weight and allowing each AS and IXP to compromise with probability 1 all virtual links passing through it. Moreover, as described by Jaggard *et al.* [14], trust policies can incorporate beliefs about a variety of other sources of network compromise, such as software vulnerabilities, physical cable tapping, and geographic location.

We do not expect that most individual users will craft their own trust policies. Indeed, doing so is likely best left to experts unless the user has especially idiosyncratic beliefs or concerns. Rather, we envision that knowledgeable specialists, such as security researchers and professionals, will provide opinions about vulnerability to specific kinds of adversaries, and that institutions, such as governments and consumer advocacy groups, will incorporate these opinions into trust policies that are appropriate for their communities. An important special case of this is that we expect the Tor Project would select a *default* policy that is in the broad interest of all Tor users, and then would configure the standard Tor client to use it as well as provide any necessary supporting data through the Tor network, much as Tor *consensuses* (*i.e.*, hourly documents describing available relays) are distributed today by a set of directory authorities and certain Tor relay mirrors.

We will consider two specific trust policies in our analysis of TAPS: (i) The *Man*, which models a single powerful global adversary whose exact location isn’t known with certainty, and (ii) *Countries*, which models each country as a potential

adversary whose locations are known exactly. Either of these policies constitutes a plausible default policy as well as that of a particular user community. We now describe these models in detail.

B. The Man

The Man represents a powerful adversary who may create, compromise, or coerce the diverse entities that make up the Tor network. Specifically, we give The Man an independent probability to observe each Tor relay family, AS organization, and IXP organization. A relay self-identifies its family in its *descriptor* [15], and an AS or IXP organization is identified using public information as being controlled by the same corporate or legal entity [11], [16]. Without any basis for differentiation, The Man compromises each AS and IXP organization independently with probability 0.1. For relays, we consider that trust may increase the longer a given relay has been active. This will not guarantee protection against an adversary that is willing to contribute persistent high levels of service to the Tor network. But it can require adversaries to either make their own persistent commitments to the network or to have effectively attacked others who have done so (and are thus most committed and experienced at participating in the network). For The Man, we thus assume each family is compromised by the adversary independently with probability between 0.02 and 0.1, where the probability increases as the family’s longevity in Tor decreases. We calculate longevity as follows: First, relay uptimes are calculated as the exponentially-weighted moving average of the relay’s presence in a consensus with Running, Fast, and Valid flags using decay parameter 0.99903776, which gives a measurement half-life of 30 days. A relay family’s uptime is simply the sum of its relays’ uptimes. The probability that a family is compromised is then taken as $(0.1 - 0.02)/(\text{family_uptime} + 1) + 0.02$.

C. Countries

As an alternative to The Man, the Countries trust policy includes as an adversary each individual country in the world. A particular country adversary compromises with probability 1 every AS or IXP that is located in that country and no others. All country adversaries are given a weight of 1. This policy illustrates a novel geographic perspective for Tor security, and it also demonstrates how we handle multiple adversaries.

III. SECURITY METRICS

A. Adversary model

As we have described in our trust model, we are considering an adversary who may control or observe some Tor relays and parts of the Internet infrastructure. Note that an important special case of this is that the adversary might observe the destination itself. From these positions, we then analyze the adversary’s success in deanonymizing users via the following methods: (i) performing a first-last correlation attack, (ii) identifying the relays used on an observed connection, and (iii) observing Tor connections over time and linking them as belonging to the same user.

As described earlier, first-last correlation attacks are possible whenever the adversary is in a position to observe traffic between the client and entry guard as well as between the destination and exit. In such a situation, we assume that the adversary can immediately determine that the observed traffic is part of the same Tor circuit and thereby link the client with its destination.

Even when the adversary is not in a position to perform a first-last correlation attack, he still may observe different parts of the circuit and use traffic correlation to link together those parts. In such a case, if the observed relays on the circuit are unusually likely for a particular client to have chosen (*e.g.*, because of atypical trust beliefs), then the adversary may be able to identify the client even without direct observation. This is even more of a concern if the adversary applies congestion or throughput attacks [9], [17] to *indirectly* identify the relays on a target circuit. Therefore, we will consider the ability of the adversary to identify the source and destination of an observed circuit based on knowledge of its relays.

Finally, it is important to consider multiple connections over time instead of just one in isolation. Every circuit that a client creates may give the adversary another opportunity obtain a sensitive position or may leak more information about the client. This problem is made worse by the fact that the adversary may be able to determine when some circuits are created by the same client. This could happen, for example, if the adversary repeatedly observes traffic to the destination and the client interacts with the same online service using a pseudonym or requests a sequence of hyperlinked documents. Observe that in both of these examples the linking is done using similarities in the content of traffic and not via any weakness in the Tor protocol itself. Thus we will consider an adversary who can link together observed connections by client.

Note that we are not considering an adversary that can identify traffic content based only on the timing and volume of data, that is, an adversary that can perform *website fingerprinting* [18]. Also, in reality we can expect *adaptive* adversaries who continually learn and shift the allocations of their resources, but we only analyze static adversaries in this paper. However, adaptiveness can be captured to a certain extent already by defining trust policies with respect to adversary behavior over time. That is, the compromise probability for a relay or virtual link can represent the probability that it will *at some point* during a given period be observed by the adversary.

B. Anonymity Metrics

We evaluate anonymity using two kinds of metrics. The first kind will give empirical estimates of the speed and frequency of first-last correlation attacks. The second kind will provide worst-case estimates for the adversary’s ability to identify the source or destination of streams that are only partially observed.

First-last correlation attacks are relatively simple and result in complete deanonymization, and therefore we are interested in accurate estimates for how likely they are to occur. Moreover, their success depends on the behavior of the user and of the circuit-creation algorithm, and therefore we can measure it empirically via simulation. Following Johnson et al. [11], we use the following as metrics:

- 1) The probability distribution of time until a client is deanonymized via a correlation attack
- 2) The probability distribution of the fraction of streams that are deanonymized via a correlation attack

Measuring the anonymity of multiple connections that are only partially observed is more difficult because it isn’t clear how successful the adversary can be at both linking separate streams and indirectly identifying relays on a circuit. Therefore, we take a worst-case approach and consider the

adversary’s ability to guess the source (resp. destination) of a sequence of streams that have been linked as coming from the same client (resp. going to the same destination) and for which the circuit relays have been identified. We measure this ability as the posterior distribution over network locations. Note that we do not take into account the fact that the adversary knows that streams for which the client or destination is unknown can only travel over virtual links that the adversary does *not* observe. Ruling out certain virtual links is more challenging than just positively identifying traffic on a virtual link because it requires the false negative rate for traffic correlation to be extremely low (in addition to the existing requirement that the false positive be extremely low). Thus, we leave this extension to our analysis to future work.

IV. TRUST-AWARE PATH SELECTION

A. Overview

We describe two variants of the Trust-Aware Path Selection algorithms (TAPS): (i) TrustAll, which is intended for system-wide deployment, and (ii) TrustOne, which works with Tor’s existing path-selection algorithm and is intended for use by a minority of users. Two TAPS variants are needed to blend in with two different types of other users: those who do use trust in path selection and those who do not. The main approach of both algorithms is to choose guards and exits to avoid first-last correlation attacks while also blending in with other users. Parts of this approach are shared with some previously-proposed path-selection security improvements [3], [4], [8]. However, TAPS includes several novel features that improve the security and performance issues of these proposals. We highlight those features before proceeding to describe the algorithms in detail.

First, TAPS uses an API that encapsulates flexible trust policies. These trust policies support adversaries from a very general class of probabilistic models. As previously described, this class can represent features such as uncertainty, multiple adversaries, and adversarial control of diverse network elements.

Second, TAPS clusters client location and (separately) destination locations. Each location cluster has a representative, and all locations in the cluster are treated as if they were the representative. The main purpose of this clustering is to eliminate the leakage of location information over time that occurs when paths are selected differently for each pair of client and destination locations. Treating all members of the cluster as if they were the representative maintains anonymity within the cluster. A secondary purpose is to reduce the amount of information needed to represent trust policies by reducing the number of paths to and from guards and exits that need be considered.

Third, TAPS treats its set of entry guards *collectively*, that is, in a way that provides security when multiple circuits, potentially using different guards, are considered. TAPS chooses each additional guard in a way that minimizes the additional exposure of its entry paths. Moreover, once a set of entry guards is chosen, TAPS doesn’t prefer one guard in the set over another when creating a connection to a given destination. This is an important design choice that prevents an adversary from observing the exit path on one connection and then deanonymizing the user by triggering a new connection to a destination for which an observable guard is preferred by the client. Tor is moving to a single guard by default to protect

against malicious guards [19]. However, multiple entry guards may still be used by clients that choose a non-default parameter setting, for example, to reduce the speed variance among circuits. Tor may also easily increase the default number of guards at any time in the future, for example, to adopt proposed enhancements that use multiple guards [20].

Fourth, TAPS provides a configurable tradeoff between security and performance by parameterizing how much relay-selection deviates from ideal load-balancing. The Tor network is under heavy load relative to its capacity [21], and latency is dominated by queuing delay at the relays [22]. Thus good load balancing is essential to maintaining Tor’s performance, which is itself a crucial factor in Tor’s success.

B. Trust API

The TAPS path-selection algorithms work with the trust policies described in Sec. II via a high-level API. Jaggard *et al.* [14] describe how to represent such policies with a Bayesian network. However, such a representation may not be the most efficient for the computations needed during path selection. Therefore, we abstract those computations into an API, and we describe how they can be efficiently implemented for The Man and Countries policies in Sec. IV-E. We assume that the API is implemented by the creator of the trust policy.

Several API functions take as an argument a network *location*. There are several possible levels of granularity at which a network location may be defined in TAPS, such as the Autonomous-System level or the BGP-prefix level. Using more fine-grained locations will result in more accurate predictions about the adversaries’ locations and thus improve security, but it will also result in increased runtime for the TAPS algorithms (and likely for the API functions as well).

We assume that API users have access to a *locations* data structure that (i) allows the locations to be enumerated, (ii) includes each location’s popularity rank for Tor clients, (iii) allows IP addresses to be mapped to locations, and (iv) includes the number of IP addresses contained in each location (e.g., the number of IP addresses originated by an Autonomous System), denoted as *l.size* for location *l*.

We also assume that API users can access a *relays* data structure that (i) allows relays to be enumerated by unique identity keys (e.g., as represented by fingerprints [15]), (ii) includes the data in each relay’s consensus entry (e.g., the status flags, weight, and IP address), and (iii) includes the data in each relay’s descriptor (e.g., the exit policy).

The trust API functions are as follows:

- **LOCATIONDISTANCE(*loc*₁, *loc*₂, *relays*)**: This function returns an abstract *distance* between two locations that measures the dissimilarity of the adversaries that appear on the network paths between the locations and the relays. This distance is the expected sum over *relays* of the total weight of adversaries that appear on one of the virtual links between the relays and *loc*₁ and *loc*₂ but not the other. This value defines a metric on the set of locations.
- **GUARDSECURITY(*client_loc*, *guards*)**: This function returns a security *score* for the use of the given *guards* as entry guards by a client in location *client_loc*. The score must be in [0, 1], and it should represent the expected total weight of adversaries *not* present on the paths between *client_loc* and *guards*, normalized by the sum of all adversary weights. Thus a higher score indicates higher security.
- **EXITSECURITY(*client_loc*, *dst_loc*, *guard*, *exit*)**: This

function returns a security score for the use of *guard* and *exit* by a client in location *client_loc* to connect to a destination in *dst_loc*. The score must be a value in $[0, 1]$, and it should represent the expected total weight of the adversaries that either are not present on the path between *client_loc* and *guard* or are not present on the path between *dst_loc* and *exit* (i.e., those not able to perform a correlation attack), normalized by the sum of all adversary weights. Thus, again, a higher score indicates higher security.

C. TrustAll

TrustAll consists of two separate processes:

- 1) **CLUSTER**: This process is run by the trust-policy provider (e.g., by the Tor directory authorities for the default policy). It clusters client and destination locations and makes the results available to clients. To maintain the anonymity sets provided by the clusters, this process should execute infrequently (e.g., every 6 months) and only to reflect significant changes in the trust on entry and exit virtual links. It takes the *locations* and *relays* data structures as inputs and produces *clusters* as output, which is a mapping from each location chosen as a cluster representative to the set of locations in its cluster.
- 2) **CONNECT**: This process is run by a Tor client. It runs every time a new connection is requested. It uses the output of the CLUSTER process, the state of the client (e.g., current network consensus, current circuits, and client IP address), *locations*, and *relays*. It may create a new circuit or reuse an existing one.

We now detail these processes.

1) **CLUSTER**: Network locations are clustered twice. One clustering will be applied to the destination's location during path selection, and the other will be applied to the client's location. The output of a clustering is a partition of network locations with a single member of each cluster in the partition designated as that cluster's representative. Client and destination clusterings are performed slightly differently because a single client is likely to visit many destinations. Therefore, if we were to bias destination clusters, we could potentially reduce security for all clients on at least one connection, but we can bias client clusters towards the most likely locations and improve security for most clients.

a) *Clustering destination locations*: Destinations are clustered with a k-medoids algorithm [23], modified to produce balanced-size clusters. Balance is needed for good anonymity, as each cluster is an anonymity set. The medoids of the resulting clusters are used as representatives. The destination-clustering algorithm takes two parameters: (i) *num_clusters*, the number of clusters to produce, and (ii) *max_rounds*, the maximum number of assignment rounds. The clustering is accomplished as follows:

- 1) Choose as an initial cluster representative a location uniformly at random from *locations*.
- 2) Choose the remaining *num_clusters*−1 cluster representatives by iteratively choosing the location with the largest distance to the representatives already chosen (i.e., the maximin distance), with distances determined by `LOCATIONDISTANCE()`. Let *reps* be all selected clusters representatives.
- 3) Assign relays to cluster representatives with `BALANCEDASSIGNMENT(locations, reps, relays)` (see Alg. 1).

- 4) Recalculate the cluster representatives by determining the location in each cluster with the smallest average distance to each other member in the same cluster.
- 5) Repeat from step (3) if any cluster representative changed and there have been fewer than *max_rounds* assignment rounds.
- 6) Return both the clusters and their representatives.

Algorithm 1 TrustAll assignment to representatives

```

function BALANCEDASSIGNMENT(locs, reps, relays)
   $A \leftarrow \text{locs} \setminus \text{reps}$  ▷ Locations to assign
  for all  $r \in \text{reps}$  do ▷ Initialize clusters and their sizes.
     $\text{clusters}[r] \leftarrow \{r\}$ 
     $\text{sizes}[r] = r.\text{size}$ 
  while  $A \neq \emptyset$  do ▷ Add closest relay to smallest cluster.
     $S \leftarrow \underset{rep \in A}{\text{argmin}} \text{ sizes}[rep]$  ▷ Smallest clusters
     $\text{min\_pairs} \leftarrow \underset{l', r' \in A \times S}{\text{argmin}} \text{ LOC DIST}(l', r', \text{relays})$ 
     $l, r \xleftarrow{R} \text{min\_pairs}$  ▷ Random closest assignment
     $\text{clusters}[r] \leftarrow \text{clusters}[r] \cup \{l\}$ 
     $\text{sizes}[r] \leftarrow \text{sizes}[r] + l.\text{size}$ 
     $A \leftarrow A \setminus \{l\}$ 
  return clusters

```

b) *Clustering client locations*: Client clustering uses known popular client locations as cluster representatives and then clusters the remaining locations in one round. The client-clustering algorithm takes as input *num_clusters*, the number of clusters to produce. It creates *reps*, the set of cluster representatives, by selecting the *num_clusters* most-popular client locations. The remaining locations are assigned to clusters with `BALANCEDASSIGNMENT(locations, reps, relays)` (see Alg. 1).

2) **CONNECT**: The CONNECT process is invoked when a Tor client is requested to connect to a destination. We assume for now that any needed DNS resolution has been performed, and CONNECT has been given a destination IP address. Section IV-F discusses how DNS resolution might occur.

The essential mechanism that TrustAll uses to improve security is to compute security *scores* for relays in the guard and exit positions and then to only select the highest-scoring relays for those positions. Guard security scores are determined with `GUARDSECURITY()`, which takes into account any existing guards when choosing a new one and thus provides security with respect to the entire guard set. Exit security scores are determined with `EXITSECURITY()`, which takes into account the guard to be used for the circuit and thus can mitigate first-last correlation attacks.

Given scores for relays in position $p \in \{g, e\}$ (*g* for guards and *e* for exits), the `SECURERELAYS()` function (Alg. 2) is used to determine the *secure* relays, that is, those relays with high-enough scores to be selected for a given position. Note that in Alg. 2 `REVERSE SORT(X, Y)` sorts the entries $x \in X$ according to the values $y[x]$, and `LENGTH(X)` returns the number of entries in *X*. `SECURERELAYS()` identifies the highest score s^* . It adds to the set of secure relays all *safe* relays, that is, relays with scores very close to s^* . Then it considers the *acceptable* relays, that is, relays with scores close to s^* but not close enough to make them safe. Acceptable

Algorithm 2 TrustAll secure relays to use for position p

```
function SECURERELAYS( $\alpha_p, scores, relays, weights$ )
   $R \leftarrow \text{REVERSESORT}(relays, scores)$ 
   $s^* \leftarrow scores[R[0]]$   $\triangleright$  Maximum score
   $n \leftarrow \text{LENGTH}(relays)$   $\triangleright$  Number of relays
   $S \leftarrow \emptyset, w \leftarrow 0, i \leftarrow 0$ 
  while ( $scores[R[i]] \geq s^* \alpha_p^{su} \wedge$ 
    ( $1 - scores[R[i]] \leq (1 - s^*) \alpha_p^{sc} \wedge$ 
    ( $i < n$ ) do  $\triangleright$  Add all safe relays
     $S \leftarrow S \cup \{R[i]\}$ 
     $w \leftarrow w + weights[R[i]]$ 
     $i \leftarrow i + 1$ 
  while ( $scores[R[i]] \geq s^* \alpha_p^{au} \wedge$ 
    ( $1 - scores[R[i]] \leq (1 - s^*) \alpha_p^{ac} \wedge (w < \alpha_p^w) \wedge$ 
    ( $i < n$ ) do  $\triangleright$  Add acceptable relays
     $S \leftarrow S \cup \{R[i]\}$ 
     $w \leftarrow w + weights[R[i]]$ 
     $i \leftarrow i + 1$ 
return  $S$ 
```

relays are added in descending order of score until the desired fraction of the total bandwidth in position p is reached.

The parameters defining these sets are given as the list $\alpha_p = (\alpha_p^{su}, \alpha_p^{sc}, \alpha_p^{au}, \alpha_p^{ac}, \alpha_p^w)$. α_p^{su} and α_p^{sc} are used just for safe relays, and α_p^{au} and α_p^{ac} are used just for acceptable relays. Safe and acceptable relays are defined using the same method, but acceptable relays use less restrictive parameters, with $\alpha_p^{au} \leq \alpha_p^{su}$ and $\alpha_p^{ac} \geq \alpha_p^{sc}$.

The “uncompromised” parameter $\alpha^u \in \{\alpha_p^{su}, \alpha_p^{au}\}$ is used to include a relay only if it has a security score s such that $s \geq s^* \alpha^u$. It must be that $\alpha^u \leq 1$, or no relays would qualify. α^u is thus the required fraction of the maximum possible expected weight of adversaries with respect to whom the circuit position is considered uncompromised. One effect of this constraint is that relays with completely untrusted paths will not be chosen if there is at least one other option.

The “compromised” parameter $\alpha^c \in \{\alpha_p^{sc}, \alpha_p^{ac}\}$ is used to include a relay only if it has a score s such that $1 - s \leq (1 - s^*) \alpha^c$. It must be that $\alpha^c \geq 1$, or no relays would qualify. α^c is thus a limit on the multiple of the minimum possible expected weight of adversaries to whom the circuit position is considered compromised. An effect of this constraint is that if relays with completely trusted paths are available, then no other options are considered.

α_p^w represents the desired minimum bandwidth fraction of relays in position p for the secure relays. It will be reached if the safe and acceptable relays together constitute at least that fraction.

Let $client_rep$ be the representative location for the client’s cluster and dst_rep be the representative location for destination’s cluster. The CONNECT process proceeds as follows:

- 1) If the number ℓ of selected and responsive guards is less than the number k desired (e.g., k is the value of NumEntryGuards [15]), then new guards are selected with CHOOSEGUARDS($\alpha_g, cur_guards, all_guards, weights, client_rep, k - \ell$) (Alg. 3), where α_g contains the security parameters for guard selection, cur_guards is the set of currently-selected guards, all_guards contains all potential guards according to the criteria Tor currently uses (e.g., sufficient uptime and bandwidth [15]), and $weights$

contains the relays’ weights for the guard position. Note in CHOOSEGUARDS() that RANDOMSAMPLE(X, Y) returns element $x \in X$ with probability proportional to $Y[x]$, and thus CHOOSEGUARDS() makes a bandwidth-weighted random selection from among the secure guards.

Algorithm 3 TrustAll additional guard selection

```
function CHOOSEGUARDS( $\alpha_g, cur\_guards, all\_guards,$ 
   $weights, rep, j$ )
   $G \leftarrow cur\_guards$   $\triangleright$  Copy of guard set to update
  for  $i \leftarrow 1 \dots j$  do  $\triangleright$  Select  $j$  new guards
     $P \leftarrow all\_guards \setminus G$   $\triangleright$  Possible guards
    for all  $g \in P$  do  $\triangleright$  Score guards for next selection
       $scores[g] = \text{GUARDSECURITY}(rep, G \cup \{g\})$ 
     $S \leftarrow \text{SECURERELAYS}(\alpha_g, scores, P, weights)$ 
     $G \leftarrow G \cup \text{RANDOMSAMPLE}(S, weights)$ 
return  $G$ 
```

- 2) Consider the existing circuits in reverse order of the time a stream was most-recently attached. If current circuit c is too *dirty*, that is, a stream was first attached too long ago (e.g., as configured by MaxCircuitDirtiness [24], which has a default value of 10 minutes), then proceed to the next circuit. Otherwise, let g_c be the circuit’s guard, let α_e contain the security parameters for exit selection, let $exits$ contain all potential exits for the desired connection according to the criteria Tor currently uses (e.g., a compatible exit policy), and let $weights$ contain the relays’ weights for the exit position. Let scores contain the exit scores, with $scores[e] = \text{EXITSECURITY}(client_rep, dst_rep, g_c, e)$ for all $e \in exits$. Compute the set of secure exit relays $S = \text{SECURERELAYS}(\alpha_e, scores, exits, weights)$. If the circuit’s exit e_c is in S , then reuse the circuit. Otherwise, proceed to the next circuit.
- 3) If no suitable circuit has been found and reused, let c be the circuit among those that are not too dirty that most recently had a stream attached, and let g_c be its guard. Choose a new exit e with CHOOSEEXIT($\alpha_e, g_c, exits, weights, client_rep, dst_rep$) (Alg. 4). Reuse c through its first two

Algorithm 4 TrustAll exit selection

```
function CHOOSEEXIT( $\alpha_e, g, exits, weights, rep_1, rep_2$ )
  for all  $e \in exits$  do  $\triangleright$  Score exits
     $scores[e] = \text{EXITSECURITY}(rep_1, rep_2, g, e)$ 
   $S \leftarrow \text{SECURERELAYS}(\alpha_e, scores, exits, weights)$ 
return  $\text{RANDOMSAMPLE}(S, weights)$ 
```

hops but “splice” e onto the end after the middle relay. This effectively operates as a new circuit, but the handshakes through the first two hops are not repeated to reduce the latency of creating it.

- 4) If no circuit exists that is not too dirty, create a new circuit as follows: (i) choose a guard g uniformly at random from the k selected and responsive guards, (ii) choose an exit e with CHOOSEEXIT($\alpha_e, g, exits, weights, client_rep, dst_rep$) (Alg. 4), and (iii) choose a middle node as Tor currently does given g and e (e.g., bandwidth-weighted random selection). Note that, in contrast to vanilla Tor path selection, the guard and exit are not explicitly prevented from being contained in the same /16 subnet or

relay family. Instead, the threat of entry and exit paths being observed by the same relay family or network is incorporated into the trust policy, and vulnerable paths are avoided by TAPS.

D. TrustOne

TrustOne path selection is designed to be used when most users are not using TAPS and instead are using vanilla Tor path selection. Thus slightly different behavior is required in order to fully blend in with the larger group. Also, if most users do not use trust, then more secure parameters can without impacting performance much.

As with TrustAll, TrustOne consists of a CLUSTER process and a CONNECT process. CLUSTER is performed in the same way as in TrustAll. The CONNECT process differs from that of TrustAll in the following ways:

- SECURERELAYS() doesn't use the notions of safe and acceptable relays. It simply orders relays by their score and chooses the most secure up to the desired bandwidth fraction. The TrustOne version of this function appears in Alg. 5. Note that the performance parameter is a single value (i.e., $\alpha_p = \alpha_p^w$). TrustOne doesn't use the concept of acceptable relays because it must allow exit relays to be chosen the same they are in vanilla Tor path selection, which in TrustOne will happen when $\alpha_e^w = 1$. Also, TrustOne can omit the concept of safe relays, which are designed to improve load balancing when there are many safe relays, because load balancing is less important when few users are using trust.
- Given a guard, potential exits are chosen exactly as they are in vanilla Tor path selection, including in particular the constraints preventing exits and guards from sharing a family or /16 subnet. This prevents a TrustOne user from being identified as using non-standard path selection (e.g., by a middle relay).

Algorithm 5 TrustOne secure relays to use for position p

```

function SECURERELAYS( $\alpha_p^w$ ,  $scores$ ,  $relays$ ,  $weights$ )
   $R \leftarrow \text{REVERSESORT}(relays, scores)$ 
   $S \leftarrow \emptyset$ ,  $w \leftarrow 0$ ,  $i \leftarrow 0$ 
  while  $w < \alpha_p^w$  do      ▷ Add desired fraction of relays
     $S \leftarrow S \cup \{R[i]\}$ 
     $w \leftarrow w + weights[R[i]]$ 
     $i \leftarrow i + 1$ 
  return  $S$ 

```

Note that a client can choose not to protect the fact the he is using TrustOne instead of vanilla Tor by setting a desired exit bandwidth fraction of $\alpha_e^w < 1$. He may do this when he doesn't believe that revealing his use of TrustOne will reveal his identity, and using a smaller α_e^w will improve his security against a first-last correlation attack by restricting his circuits to more-secure exits.

E. Trust API implementations

The efficiency of the trust API depends on the type of trust policies used. For example, a user with trust in individual relays may only need to store a single trust value for each relay and perform simple arithmetic computations for the API functions, while a user with trust in Autonomous Systems may need to store an entire Internet topology and perform routing

inference. In general, because the API functions return the expectation of values that can easily computed if compromised relays and virtual links are unknown, they can be implemented by repeatedly sampling the adversary distributions. Thus the API functions are compatible with the Bayesian-network representation of Jaggard *et al.* [14]. However, policies should use implementations that are efficient for their specific features.

For The Man policy, the trust API functions need access to data describing the relay families, AS organizations, IXP organizations, and the virtual entry and exit links on which each AS and IXP organization has a presence. The API functions can easily be implemented efficiently for The Man because there is a single adversary whose presence on a virtual link depends on the compromised status of *network entities* (i.e., relay families, AS organizations, and IXP organizations) that are each independently compromised. We implement the API functions as follows:

- LOCATIONDISTANCE($loc_1, loc_2, relays$): To compute this, consider each $r \in relays$. Let E_1 be the set of network entities that exist between r and both loc_1 and loc_2 , let E_2 be the set of network entities that exist only between r and loc_1 , and let E_3 be the set of network entities that exist only between r and loc_2 . Let p_r be the probability that the adversary is present on one of the paths from r to loc_1 and loc_2 but not the other. p_r can be computed as the probability that (i) no $e \in E_1$ is compromised and (ii) either some $e \in E_2$ is compromised and no $e \in E_3$ is compromised or vice versa. The distance is computed as the sum of p_r over all $r \in relays$.
- GUARDSECURITY($client_loc, guards$): Let E be the set of network entities between $client_loc$ and the *guards*. The security score computed as the product of the probabilities that each $e \in E$ is individually uncompromised.
- EXITSECURITY($client_loc, dst_loc, guard, exit$): Let E_1 be the set of network entities that exist both between $client_loc$ and $guard$ and between dst_loc and $exit$, let E_2 be the set of network entities that exist only between $client_loc$ and $guard$, and let E_3 be the set of network entities that exist only between dst_loc and $exit$. The security score is the product of the probability that no $e \in E_1$ is compromised and that either no $e \in E_2$ is compromised or no $e \in E_3$ is compromised.

For the Countries policy, the trust API functions need the list of all countries as well as a data structure mapping each relay to its country and each virtual link to the countries it passed through. LOCATIONDISTANCE($loc_1, loc_2, relays$) simply sums over $r \in relays$ the number of countries on which the virtual links $\{loc_1, r\}$ and $\{loc_2, r\}$ disagree. GUARDSECURITY($client_loc, guards$) returns the fraction of countries not containing *guards* or on the virtual links between $client_loc$ and *guards*. EXITSECURITY($client_loc, dst_loc, guard, exit$) returns the fraction of countries either not containing *guard* and not on the $\{client_loc, guard\}$ virtual link or not containing *exit* and not on the $\{dst_loc, exit\}$ virtual link.

F. Discussion

So far, we have been discussing trust-aware routing to destination IP addresses. Many or most connections will require DNS resolution before the destination IP is known. Exit relays resolve DNS requests in current Tor routing to prevent linking of client IP address directly to a destination DNS request. This must also be done in a trust-aware manner,

or there is little point in using trust-aware routing from exit to destination once the IP address is known. If we rely on a chosen exit to control the DNS resolution, then, even if it shares the default trust values, it may not be a good exit for resolving the intended destination. When doing iterative DNS resolution from the client, possibly switching to new circuits depending on the next identified DNS resolver, the performance overhead could be significant. In this paper, we assume that primary and backup nameserver ASes are included with exit-relay descriptors. Assuming that these are typically in the same AS as or immediately adjacent to the exit, this will at least make sure that initial DNS requests are trust-aware. How best to address DNS issues beyond that is outside the scope of this paper.

We expect the CLUSTER process to be performed by organizations (e.g., EFF) and the results distributed to users who trust their analysis and calculations. End-users would then only need to perform the CONNECT process. In the case of TrustAll, security depends on the assumption that many other are users using the same policy. Therefore, the TrustAll CLUSTER process could be integrated with the standard Tor release and enabled with a default policy (e.g., The Man or Countries). However, TrustOne was designed to be used by a minority of users, and while the algorithm could be included with Tor, it would not be enabled by default. We analyze the security and performance of both approaches in the following sections.

V. SECURITY ANALYSIS

A. Experimental Setup

We experimentally evaluate the security of the TrustAll and TrustOne algorithms against The Man using an Internet map, data about the Tor network, and path-selection simulators for TAPS and for vanilla Tor. We construct an AS-level Internet map using traceroute-based topology data and inferred AS relationships from CAIDA [25], BGP routing tables supplied by Route Views [26], and sibling information based on RIPE WHOIS records. Routes between AS locations on our map are inferred using the algorithm proposed by Qiu and Gao [27], which takes AS paths observed in BGP tables and extends them to other locations using shortest “valley-free” paths. We identify IXPs and place them on these paths using data from the IXP mapping project [28]. Conversion of IPs to ASes is performed using routing prefix tables from Routeviews. We group ASes into commercial organizations using the results of Cai *et al.* [16]. We conservatively group IXPs into organizations based on similarities in their listings in the Packet Clearing House and PeeringDB (see [11] for details).

We use data from Tor Metrics [21] to evaluate how the security of TAPS would have compared to vanilla Tor if it used the Tor network as it existed in the recent past. Specifically, we use the archived network consensus and server descriptors to determine such facts as which relays existed, what families they were grouped in, what their relative bandwidth capacities were, what their “exit policies” were, and which had the needed stability to serve as guards or as exits for “long-lived ports”¹. We also use data from Juen [6] to identify the 401 most popular Tor client ASes. The most popular client AS identified by Juen is AS 6128 (Cable Vision Systems, US).

We simulate path selection on past Tor networks using the Tor Path Simulator (TorPS) [11]. TorPS is based on the code in Tor version 0.2.4.23. TorPS provides Monte Carlo simulation of user circuit creation over weeks and months on the changing Tor network. Each TorPS sample consists of a sequence of circuits and assignments to those circuits of requested user connections over the period of simulation. We use TorPS unmodified to evaluate the security of vanilla Tor, and we also modify TorPS to use the TAPS path selection algorithms.

We perform our TorPS simulations for two models of user behavior: the *Typical* model, and the *IRC* model. Johnson *et al.* describe these models in detail [11]. The Typical model consists of four 20-minute user traces obtained from actual (volunteer) user activity over Tor: (i) Gmail / Google Chat, (ii) Google Calendar / Docs, (iii) Facebook, and (iv) web search activity. It includes 205 unique destination IPs and uses TCP ports 80 and 443. These traces are played every day in five sessions between 9 a.m. and 6 p.m. This results in 2632 TCP connections per week. The IRC model uses the trace of a single IRC session to `irc.oftc.net` on port 6697, which we observe to resolve to 82.195.75.116 in AS 8365 (TU Darmstadt, DE). This model repeatedly plays the trace 8 a.m. to 5 p.m. every weekday. This results in 135 TCP connections per week.

To evaluate security with respect to The Man, we use it to draw a sample of the compromised relays and virtual links for each TorPS sample and consider the security of that sampled path-selection behavior against that sampled adversary. That is, we independently assign a compromised status to each AS organization, IXP organization, and relay family using the probabilities given in Section III. We then consider the anonymity of the circuits in the TorPS sample against the sampled adversary.

We run simulations over the first week of December 2013. To the extent possible, our Internet and Tor data is taken from this period as well.

B. Location Clusters

The TrustAll algorithm only prevents the node tuples from revealing client and destination locations up to their location clusters. An adversary that can identify the relays in each position of a circuit may use it as evidence for the clusters of the client and destination. For example, if the adversary is also observing the exit-destination link, it may be the case that a given guard and exit would only be used to visit that destination by members of a given cluster. This is especially likely for an adversary that can additionally link together multiple connections as belonging to the same (pseudonymous) user. This powerful attack has not been recognized in prior work on network-location-aware path selection.

Thus we must consider the anonymity that is afforded when a client or destination is known to belong to a given cluster. In our experiments, we partition all Internet ASes into 200 clusters. This number of clusters allows for significant diversity in cluster behavior while reducing the anonymity set of roughly 3.7 billion addresses in IPv4 by a factor of 200. We perform clustering using as the guard and exit locations the sets of ASes in which Tor guards and exits were observed to reside during the six-month period from June 2013 to November 2013, which precedes the simulation period in December 2013.

Following the cluster-formation algorithm given in Sec-

¹TCP ports 21, 22, 706, 1863, 5050, 5190, 5222, 5223, 6523, 6667, 6697, and 8300

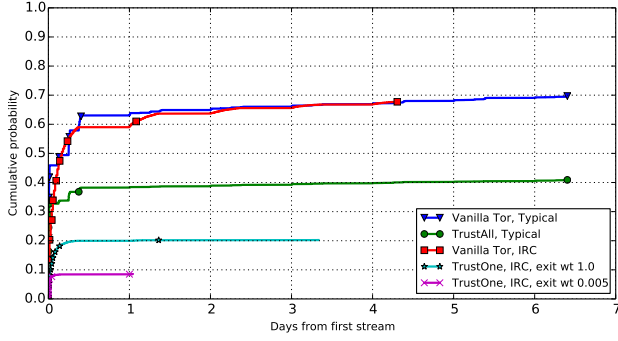


Fig. 1: Time to first compromise in The Man model

tion IV, the 200 client clusters are created by choosing the top 200 Tor client ASes as reported by Juen [6] and assigning the remaining locations based on the security they experience when using the cluster representatives’ guard and exit choices. In the resulting clustering, the median cluster size in terms of contained addresses is 11,363,072, the minimum size is 10,840,321, and the maximum size is 118,966,528. We can use the number of ASes in each cluster as a measure of location diversity within clusters. The client clusters contained a median of 171 ASes, with a minimum number of ASes of 1 and a maximum number of 900.

The destinations clusters were formed slightly differently because users typically visit many destinations, and thus most users are affected by the amount of agreement in Internet paths to Tor exits between the long tail of destinations and their cluster representatives. Thus k-means clustering was used to identify representatives that were best able to minimize distances between cluster members and their representatives, as described in Section IV. The clusters that were the output of this process had a median of 11,992,641 IPv4 addresses, with a minimum of 11,358,466 and a maximum of 119,068,928. Again, the number of contained ASes provides a measure of diversity within the clusters, and the median contained number was 159, with a minimum of 2 and a maximum of 1280.

C. TrustAll Security

First we consider security against The Man when all users use TAPS as the default path-selection algorithm (*i.e.*, TrustAll). In particular we consider the threat of complete deanonymization via first-last correlation. We used the following security parameters:

$$\alpha_g^{su} = 0.95, \alpha_g^{sc} = 2.0, \alpha_g^{au} = 0.5, \alpha_g^{ac} = 5.0$$

$$\alpha_e^{su} = 0.95, \alpha_e^{sc} = 2.0, \alpha_e^{au} = 0.1, \alpha_e^{ac} = 10.0$$

$$\alpha_g^w = 0.2, \alpha_e^w = 0.2.$$

Figures 1 and 2 show cumulative distributions for when and how often such compromise occurs for a Typical user in the most popular Tor client AS (6128) over 7 days of simulation.

We can see that TrustAll significantly reduces the chance of first-last correlation by The Man as compared to vanilla Tor. Users coming from AS 6128 see the probability of at least one successful first-last correlation attack drop from 0.7 to about 0.4. Observe that this overall reduction occurs both because the chance of choosing a compromised guard among

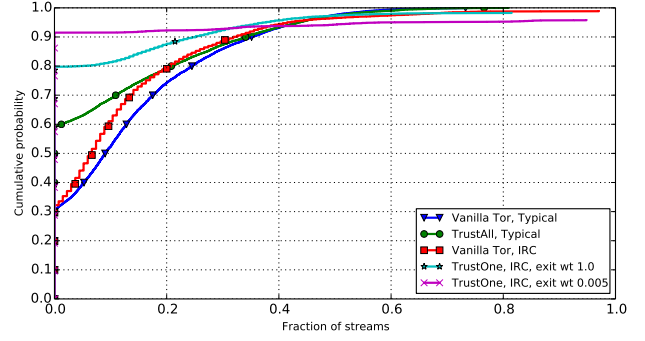


Fig. 2: Fraction of compromised streams in The Man model

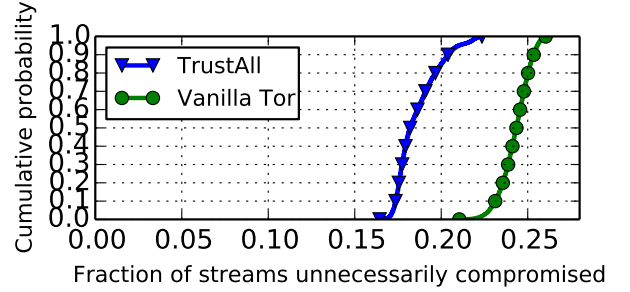


Fig. 3: Security in the Countries model

the initial set of 3 is reduced (as seen in the values at 1 day) and because the chance of choosing additional compromised guards, precipitated by network churn, is reduced (as seen in the smaller slopes of the CDF). The results also show that the median fraction of compromised streams drops from around 0.1 to 0.

Next we consider the security of TrustAll in the Countries model. In this model, users face multiple country adversaries (249), each of which deterministically compromises all ASes and IXPs within its borders. In this setting, users are sometimes necessarily compromised against those countries that contain both the source and destination AS. Thus we only consider the fraction of streams which are “unnecessarily” compromised by some country. Figure 3 shows the distribution of this value for a Typical user in the top Tor client AS (6128, located in the US) active over seven days. It shows that TrustAll reduces the fraction of unnecessarily-compromised streams from a median of about 0.24 to a median of about 0.17.

Finally, we consider security of IRC users using the TrustOne algorithm when default users are using vanilla Tor. In this case, the TAPS users choose guards and exits in a way that is sufficiently similar to how they are selected in vanilla. Specifically, guards are selected using $\alpha_g^w = 0.005$ and exits are selected using either $\alpha_e^w = 0.005$ or $\alpha_e^w = 1$. The former weight for exits results in a TrustOne user being 200 times more likely to have chosen a given exit than a vanilla Tor user. This could be an appropriate setting for a user who is not concerned with revealing his use of TrustOne and his trust beliefs. It could also be appropriate for a user who is just trying to protect anonymity for a single connection considered in isolation. The weight $\alpha_e^w = 1$ results in exit selection by

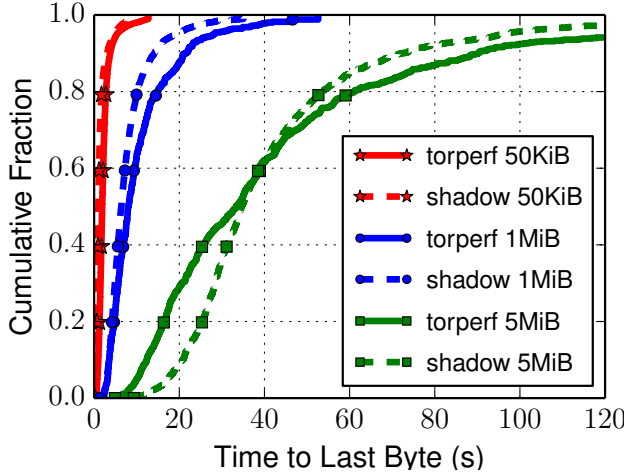


Fig. 4: Performance comparison of our private Shadow (shadow) network and the public Tor (torperf) networks

a TrustOne user that is identical to that of Tor users. This is an appropriate setting when the user wants to hide his use of TrustOne and the user’s adversaries may be able to link circuits together over time as belonging to the same user.

Figures 1 and 2 shows the chance of deanonymization of an IRC user in AS 6128 via first-last correlation for TrustOne and vanilla Tor. We can see that TrustOne results in a significantly lower chance of compromise, from a 0.68 chance for vanilla Tor users to about 0.2 or 0.1, depending on the exit-selection parameter α_e^w . The median compromise rate also drops from about 0.7 to 0.

VI. PERFORMANCE ANALYSIS

Our TAPS algorithm presented in Section IV was designed to provide tunable performance while improving users’ security by limiting the probability that an adversary may observe both sides of a Tor circuit. Having just analyzed the extent to which TAPS improves security for Tor users in Section V, we now analyze how various security configurations affect performance and the balancing of load among relays.

A. Tor Network Model

We evaluate the performance effects of TAPS using Shadow [12], [13], a scalable and deterministic discrete-event network simulator with a plug-in architecture that enables it to run real software. Using Shadow has multiple advantages over other experimentation approaches: (i) Shadow ensures safe experimentation and eliminates privacy risks by launching a large-scale private Tor network deployment on a single machine and interposing all networking functions; (ii) because Shadow is deterministic, the results we obtain through experimentation are reproducible; (iii) Shadow’s control over time and ability to isolate node behaviors enables us to explore parameters and attribute performance effects to specific configuration changes; and (iv) Shadow runs the Tor software, and so we can directly implement our algorithms in Tor’s code base while increasing our confidence that the application-level performance effects are realistic.

We configure a private Tor deployment using Shadow and the topology and configuration files distributed with Shadow. We swap the small scale Internet topology distributed with

Shadow for the much more detailed large scale topology produced by Jansen *et al.* [22]. Our base configuration consists of 400 Tor relays (4 directory authorities and 93 exits), 1380 Tor clients that also run a simple file fetching application, and 500 simple file servers. Of the clients, 1080 are “Web” clients that are configured to: choose a random server and download a 320 KiB file from it; pause for [1, 60] seconds chosen uniformly at random; repeat. 120 of the remaining clients are “bulk” clients that are configured to repeatedly download a 5 MiB file without pausing between downloads. Each experiment is configured to run for 1 virtual hour, which takes about 5 hours on our machine (using 12 Shadow worker threads) while consuming 40 GiB of RAM.

We also run 180 “ShadowPerf” clients and model their behavior after the TorPerf [29] clients that measure and publish performance over time in the public Tor network. Of our 180 ShadowPerf clients, 60 download 50 KiB files, 60 download 1 MiB files, and 60 download 5 MiB files; they all pause for 60 seconds between each download and are all configured with a MaxCircuitDirtiness of 10 seconds to ensure that each download goes over a fresh circuit. The performance results obtained with our ShadowPerf clients may then be directly compared to the performance of the public Tor network as measured by TorPerf, and provides one metric for simulation accuracy. Figure 4 shows the result of a comparison of public Tor performance to Shadow Tor performance, and indicates that Shadow is able to model Tor performance characteristics with reasonable accuracy over a range of file download sizes.

B. TAPS Implementation Details

We branched shadow [30] at commit 023055eb5, shadow-plugin-tor [31] at commit 9eed6a7c5, and Tor at version 0.2.5.2-alpha and modified them to support experimentation with TAPS. Modifications were required for Shadow and the Tor plug-in to facilitate the assignment of nodes to ASes and node lookup of the assigned AS numbers and country codes as required by TAPS. The majority of the implementation of TAPS was done inside of Tor itself, including: a “trustdb” with support for the parsing, storing, and accessing the trust information necessary for the nodes to use TAPS to select paths and build circuits (e.g., the location of relays, adversaries, and source and destination clusters); and the implementation of both the TrustOne and TrustAll versions of TAPS as described in Section IV. Our modifications resulted in 1126 additional lines of C code in Tor.

We next evaluate the performance effects of TAPS by evaluating both the TrustAll and the TrustOne variations in Shadow, and comparing the performance to that achieved by vanilla Tor’s bandwidth-weighted path selection.

C. TrustAll Against The Man

Recall that in the TrustAll variation of TAPS, all users in the network are configured to select paths based on the default trust policy. We explore the performance of TAPS under different configurations of the parameters described in Section IV. We use the same values of the parameters defining safe and acceptable relays in position $p \in \{g, e\}$ (i.e., α_p^{su} , α_p^{sc} , α_p^{au} , and α_p^{ac}) that were used for the security experiments in Section V-C. We then adjust the required bandwidth fraction $\alpha^w = \alpha_g^w = \alpha_e^w$ in order to adjust the amount of load balancing that happens due to client path selection. Higher values of α^w relax security by requiring clients to consider more nodes in an attempt to exceed the bandwidth fraction

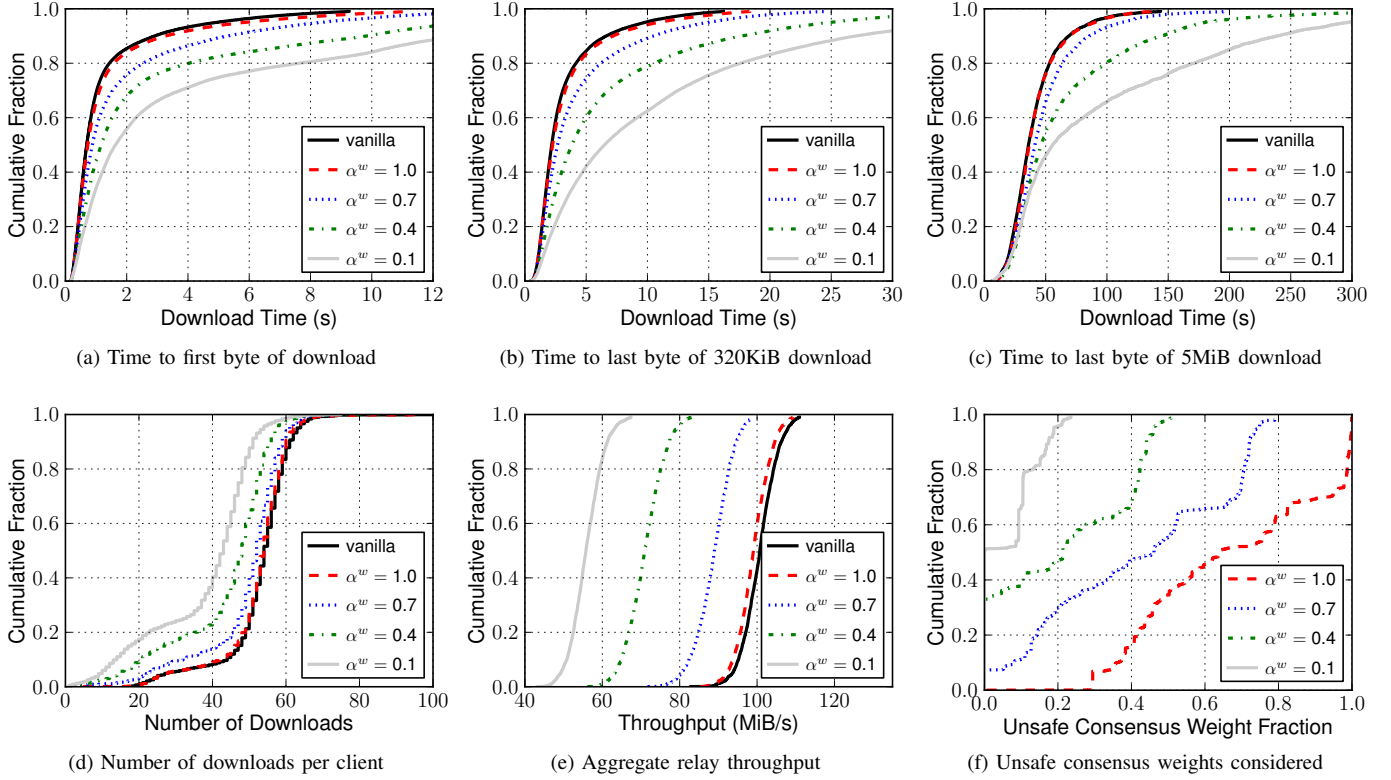


Fig. 5: Performance of the TrustAll variation of TAPS against The Man policy, varying required bandwidth fraction α^w

and allow the algorithm to better distribute load among relays that have the capacity to support it (relays that are not safe or acceptable are never chosen in any case). Lower values of α^w reduce the number of relays that a client must consider, which means they effectively prefer more secure relays and perform less load balancing. We experiment with different values of α^w to explore these effects.

The results of our experiments are shown in Figure 5. Figure 5a shows the distribution of the time to receive the first byte of each download aggregated across all clients in our network. As can be seen, there is a significant and consistent improvement in latency to the first byte as α^w increases. As α^w increases, client load is better distributed because more clients will end up choosing high capacity nodes even if they are not the most secure choice. Similar results are shown for time to complete the downloads, for Web clients in Figure 5b and bulk clients in Figure 5c. Also, performance differences are consistent across the α^w settings for both Web and bulk clients, which we would expect since our path selection algorithm is the same in both cases and we are not adjusting traffic schedulers across our experiments.

Many of our experiments resulted in a recognizable decrease in performance, and even with $\alpha^w = 1.0$, a long tail is present in the data. We expect this to be the case since any deviation from Tor’s default bandwidth-weighted algorithm will result in suboptimal load balancing. However, our results indicate that a clear performance-security trade-off is possible by using TAPS, and that the algorithm can be tuned to a desired level of performance while still removing the least secure relays from consideration.

A side effect of the decrease in performance is fewer completed downloads by each client over the course of the experiment due to our behavior models, as evident in figure 5d. Related to download times, there is a significant reduction in the number of downloads for clients (and a long neck for about 20 percent of Web clients). This is likely due to the fact that these clients, because of their location, consistently choose low capacity guards and exits that cause their downloads to receive bad performance. (Clients in the long neck of number of downloads are also in the long tail of download times.) This is also a result of our behavior models, in which clients do not start a new download until the previous one finishes. A richer behavior model in which some clients start multiple downloads at a time (e.g., representing users opening multiple tabs or starting multiple background file transfers) could alleviate this artifact.

As shown in Figure 5e, the reduction in the number of downloads also reduces total aggregate network throughput (bytes written summed across all relays every second). This again indicates a reduction in the ability of Tor to properly balance load when all clients in the network use TAPS. Again, $\alpha^w = 1.0$ performs the closest to vanilla Tor and does not result in a significant loss in performance, despite removing the least secure relays during path selection.

Finally, Figure 5f shows the cumulative fraction of bandwidth weight from relays that fall outside of the safe thresholds but that were still considered during path selection. These relays represent those that were within the acceptable thresholds but not within the safe thresholds. Recall that TrustAll selects relays in this acceptable zone one at a time, from most

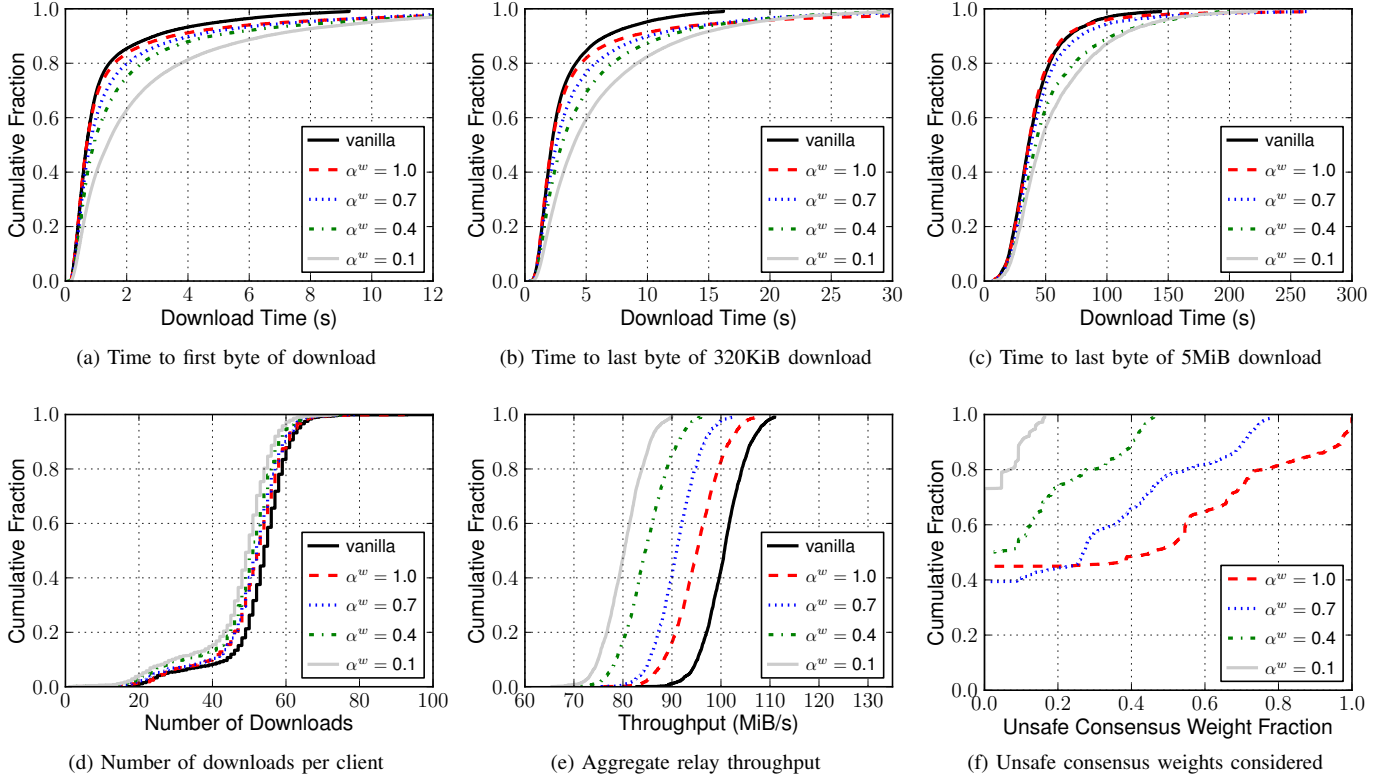


Fig. 6: Performance of the TrustAll variation of TAPS against the Countries policy, varying required bandwidth fraction α^w

to least secure, until the desired consensus weight fraction α^w is reached. As expected, the more performance that is demanded (*i.e.*, as α^w increases), the more relays outside of the safe thresholds must be used to reach the desired performance. Our results indicate that there are settings of α^w that result in performance nearly as good as Tor’s default performance-optimized algorithm, while also taking security into consideration.

D. TrustAll Against Countries

The experimental results discussed above were obtained using The Man policy. For completeness, we also experimented with the same parameters using the Countries policy. Figure 6 confirms that the same trends are present against the Countries policy as were discussed above, and the results increased our confidence in the conclusions drawn about the performance of TAPS.

E. Trading Security for Performance

Figure 7 demonstrates how TAPS directly trades performance for security according to the parameter α^w . Figure 7a shows the security-performance tradeoffs of TrustAll against The Man policy for various values of α^w . Shown in the figure are two performance metrics: “Client Download Time” represents the median across all clients of the median download time for each client; “Relay Throughput” represents the median application throughput in terms of bytes written per second, across all relays over all seconds during the experiments. Both of these metrics are normalized with respect to vanilla Tor, meaning that values closer to 1.0 indicates that TAPS achieves performance more similar to that achieved by vanilla Tor. Also

shown in Figure 7a are the “Probability of Compromise” and the “Stream Compromise Rate” as metrics of security. The metrics are again normalized with respect to vanilla Tor, so that values closer to 0 are less similar to vanilla Tor and indicate higher security. As is clear in the figure, as the tradeoff parameter α^w increases, both of the performance metrics improve while both of the security metrics get worse. This is expected: as more relays are utilized to reach the performance requirements of α^w , it is more likely that insecure relays or relays that exist on insecure paths will be selected and used in a circuit.

A similar analysis applies to the Countries policy, the results for which are shown in Figure 7b. The security metrics include the median fraction of “Unnecessarily Compromised Streams”, where the source and destination of a stream do not exist in the same country and yet the stream was still compromised, and the median number of countries with which the client unnecessarily had a compromised circuit. The performance metrics are as above. The same basic trends hold for the Countries policy: as α^w increases and the number of potentially unsafe relays considered for a path increases, so does the number of avoidable stream compromises and the number of countries to which a given client is unnecessarily compromised. In all cases, however, security improves with respect to vanilla Tor while performance decreases depending on the tunable setting of the tradeoff parameter α^w .

F. TrustOne Against The Man

In order for TrustAll to be effective, most clients must use it. If only a minority of clients use trust, then they should use

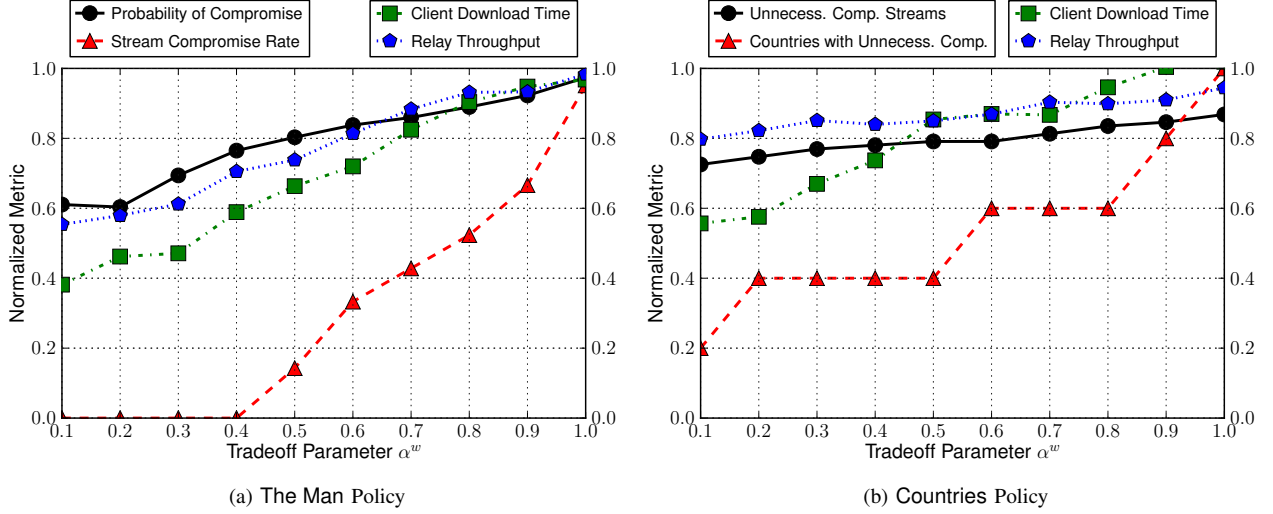


Fig. 7: Trading of performance and security in TrustAll with α^w

metric	$\alpha^w = 0.005$	$\alpha^w = 1.0$	vanilla
Time to First Byte	0.870, 1.548	0.783, 1.694	0.690, 1.419
Time to Last Byte 320KiB	3.806, 3.785	2.685, 3.255	2.172, 2.597
Time to Last Byte 5MiB	39.825, 29.342	35.203, 14.395	35.777, 20.658
Tor Throughput (MiB/s)	98.635, 4.893	99.699, 5.387	100.660, 4.250

TABLE I: Statistical summary (median, standard deviation) of performance for TrustOne, when a small set of clients use trust while attempting to blend in with Tor’s default bandwidth-weighted path selection algorithm

TrustOne in order to blend in with vanilla-Tor users. They can also take advantage of their minority status by using higher-security parameters without affecting Tor’s load balancing much.

We demonstrate the performance of TrustOne by configuring 68 of our Web clients and 5 of our bulk clients to run the TrustOne algorithm with $\alpha_g^w = 0.005$ and $\alpha_e^w \in \{0.005, 1.0\}$; the other parameter settings are as in the TrustAll experiments. All other clients use the vanilla Tor path-selection algorithm. Thus the TrustOne clients choose a very secure guard, and they either choose exits identically to vanilla-Tor users in order to blend in ($\alpha_e^w = 1.0$) or don’t attempt to hide their use of TrustOne and instead choose exits very securely ($\alpha_e^w = 0.005$).

Figure 8 compares the performance of the vanilla and trust clients, and Table I provides a statistical summary. Note that the results for $\alpha^w \in \{0.005, 1.0\}$ come from two separate TrustOne experiments and the vanilla results come from another experiment with no TrustOne clients. Also note that the reported download times for $\alpha^w \in \{0.005, 1.0\}$ are only for TrustOne clients. Across all three client performance metrics (time to first byte, and time to last byte of Web and bulk downloads), we see only a very small drop in client performance for both settings tested. Although our sample size is small, both settings of α_e^w resulted in similar performance for the trusted user set. This indicates that performance for those clients was due to the capacity and congestion of their guard nodes (which they chose using a very secure value of

α_g^w). Also shown in Table I are results showing that relay throughput in the TrustOne experiments was not significantly lower than in the vanilla Tor experiment (relay throughput is over all relays and thus in the TrustOne experiments includes traffic from both trust-aware and vanilla clients). This is attributable to a relatively small change in the load balancing across the network since only the trust users deviate from the optimized load balancing algorithm. Our results indicate that little performance is lost from using the TrustOne algorithm when a relatively small set of users are doing so.

VII. TRUST ERRORS

Because the client’s paths depend on her beliefs and may not be accurate, it is important to investigate the effects of errors in the client’s beliefs on her security. Here, we do that experimentally by considering a variety of mismatches between client trust beliefs and the actual adversary. We look at three client behaviors against nine different actual adversaries for a single location (AS 6128) over one week. We also look at our Typical client in 401 different locations with trust beliefs corresponding to The Man, but where the actual adversary distribution is one of a selected set of other behaviors.

The client might also have beliefs about the structure of the network. Errors in those may have significant impacts on the client’s security if, for example, the client believes that an untrustworthy AS organization does not contain an AS that it actually does. We focus our experiments here on errors in trust beliefs.

We consider three different client behaviors: The Typical and IRC clients are as described above. We also consider a client with Typical network behavior who chooses paths based on trust beliefs that match the Countries adversary.

These client properties are combined with various adversaries, which may or may not match the client’s beliefs if she is using TAPS:

- Type 0** The adversary is The Man adversary described above.
- Type 1** The probability of compromise is increased, relative to The Man, by a factor of 1.25 for AS/IXP organizations, lone ASes, and relay families. This reflects the possibility that the client uniformly underestimates the adversary’s capabilities.

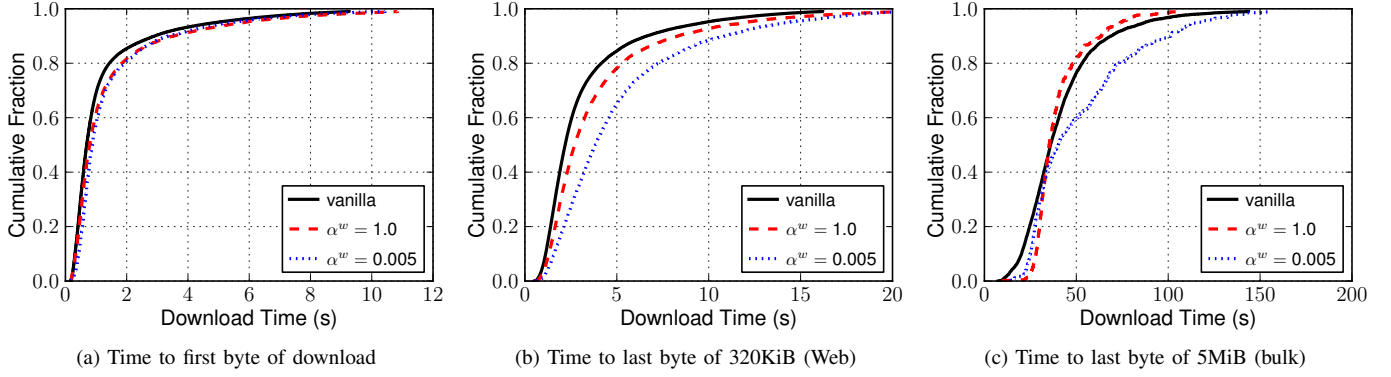


Fig. 8: Performance comparison of the trust-one variation of TAPS for various security settings

Type 2a This is the same as The Man except ASes that are not part of any organization are not compromised.

Type 2b This is the same as The Man except ASes that are not part of any organization are compromised with probability 0.05.

Type 3 For each run, half of the AS organizations and half of the IXP organizations are compromised with probability 0.15, and the others are compromised with probability 0.05. For efficiency, an AS that is not part of an AS organization is only assigned a compromised status when it is first encountered on a virtual link during analysis. Upon its initial observation, such an AS is assigned one of 0.15 and 0.05 as a compromise probability using a fair coin, and then it is compromised with that probability.

Type 4 The adversary is the same as The Man except longer uptime *increases* the compromise probability for a relay family (e.g., because of the increased chance of unpatched software). In particular, the compromise probability for a relay family with uptime t_f is $0.1 - (0.1 - 0.02)/(t_f + 1)$.

Type 5 The adversary compromises each relay with probability 0.1 and each virtual link with probability $0.3439 = 1 - 0.9^4$. (The latter value is chosen to approximate the probability of compromising ASes/IXPs independently. On the virtual links that we consider, the median number of ASes/IXPs is four, although these are not necessarily from distinct organizations.)

Type 6 The adversary is the same as The Man for ASes/IXPs. For relays and relay families, the adversary compromises nontrivial families with probability 0.1 and individual relays that are not part of a nontrivial family is 0.05.

Type 7 The adversary is the same as The Man for ASes/IXPs. For relays and relay families, the adversary compromises families with probability $p_{max} - (p_{max} - p_{min})2^{-(f_{size}-1)}$, where p_{min} and p_{max} are the minimum (0.02) and maximum (0.1) probabilities of family compromise for The Man and f_{size} is the number of relays in the family.

Table II shows the median time to first compromise (TTFC) in days and the probability that some circuit is compromised for the three different client types and nine different adversary distributions noted above. In each case, we take the client to be in AS 6128. The data are obtained from 10,000 simulations of client behavior from 12/1/13 to 12/7/13. Values of “> 7” for the TTFC indicate that the value is at least seven days, the length of these experiments.

Table III shows various compromise statistics for a Typical

Client: Typical, against The Man									
Adv. →	0	1	2a	2b	3	4	5	6	7
TTFC	> 7	> 7	> 7	> 7	> 7	> 7	.01	> 7	> 7
Prob.	.4	.49	.4	.4	.41	.47	.68	.47	.47
Client: IRC, against The Man									
Adv. →	0	1	2a	2b	3	4	5	6	7
TTFC	> 7	4.17	> 7	> 7	> 7	> 7	.07	> 7	> 7
Prob.	.41	.5	.41	.41	.41	.48	.71	.49	.48
Client: Typical, against Countries									
Adv. →	0	1	2a	2b	3	4	5	6	7
TTFC	.38	.01	.38	.38	.38	.26	.0	.25	.38
Prob.	.58	.66	.56	.57	.57	.6	.82	.61	.58

TABLE II: Median time to first compromise (in days) and probability of compromise for three different client behaviors and nine different actual adversary distributions. Data are from 10,000 simulations of a client in AS 6128 running for seven days.

Adv.	Med. TTFC			Med. Prob.			Med. Frac.		
	min./med./max.			min./med./max.			min./med./max.		
2a	.01	> 1	> 1	.21	.45	.66	.0	.0	.09
4	.01	> 1	> 1	.27	.49	.69	.0	.0	.11
5	.0	.01	.01	.59	.66	.79	.1	.13	.16

TABLE III: Statistics (minimum, median, and maximum) on the median times to first compromise (in days), compromise probability, and fraction of paths compromised over 10,000 trials for each of 401 client locations running for one day each against a selected set of adversaries. The clients choose paths against The Man; the actual adversary is shown in the first column.

client who chooses paths based on beliefs that match The Man against three different adversary distributions. For each of 401 client locations, we ran 10,000 simulations and took the median TTFC, compromise probability, and fraction of compromised paths for that location. The table shows the minimum, median, and maximum of these per-location median values. Values of “> 1” for the TTFC indicate that the value is at least one day, the length of these experiments.

Comparing Table II with Fig. 1, we see that when the client in AS 6128 chooses paths against The Man, the use of TAPS increases her security, compared with vanilla Tor, against adversaries that are related to The Man even though the

client is wrong about the exact nature of the adversary. More precisely, this is true for all of the adversary types considered in this section other than Type 5, which is the adversary that independently compromises each relay and virtual link and is the only type that does not compromise the network based on organizations and families. When the adversary is actually of Type 5, Tables II and III show that it is able to do quite well against the client over many locations and client behaviors.

VIII. OBTAINING AND PROPAGATING TRUST

In order to use trust to make decisions about route selection, clients require either adequate trust information themselves or the results of trust-aware routing algorithms performed on their behalf.

For a widely-used trust-aware Tor network supporting users with diverse trust beliefs we may need to consider propagating trust via directory systems with controls on what trust information is distributed to whom and controls on what the directory servers know about what they are distributing. For now, we simply ask what basic information needs to be passed to clients to support trust-aware route selection as we have been describing in this paper.

We will assume that default relay trust information is distributed via the same directory and distribution system that Tor currently uses to distribute information about relays and their state. This information could be gathered by relays themselves and signed. It would then make sense to have this distributed via directory authorities. A compromised exit relay already sees the destination of connections passing through it. However, by lying about the path from itself to various destinations, a relay could conceivably attract more traffic from TAPS-routed circuits. More troubling, by gaming the trust system in this way relays could attract traffic intended for specific targeted destinations. For this reason it might be advisable to have others determine, or at least somewhat verify, the announced routes between exits and destinations. That problem has been studied in many other settings (see, *e.g.*, the surveys [32], [33]); we leave the addressing of that issue in this context to future research.

We assume for current purposes that clients that are not using default trust information will have obtained it separately in some way. For example, workstation and mobile clients may have received it over a private LAN from whatever authority provides them their trust information.

We consider as an example how much data must be stored to implement *The Man* policy. First, the client must be able to determine its cluster. There are 46368 ASes in our network map, and so at most two bytes are needed to represent each AS. With 200 client-cluster representatives, 400 bytes are needed to represent all possible AS adversaries for all client cluster representatives. Thus 182 KiB suffice for each client to know its cluster.

Second, to choose guards, clients need to know the longevity of guards, which they can get from the consensus, and they need the ability to calculate the joint probability that adversaries will appear on any virtual link from their cluster representative to members of a candidate guard set. For data gathered December 2013, all guards are within 603 ASes, and the average number of AS or IXP organizations on a virtual link between client representative and guard is 4.044. At two bytes per adversary (there are only 359 IXPs), an explicit listing of the relevant adversaries on all virtual links for a

given client cluster representative is $603 \times 4.044 \times 2 = 4.7\text{KiB}$. Thus the total amount of information that clients need to calculate their guards is less than 190 KiB (additional to what they would have already included from a present-day Tor consensus).

Third, on the destination end, clients need to be prepared to route to any destination cluster representative, and the amount of information needed commensurately greater. To choose exits, users need to obtain all the relevant information to apply the *The Man* policy for each link between an exit and a destination cluster. Again for data gathered December 2013, all exits (any relay allowing connection to *some* IP and port) are contained within 962 ASes (all guards and exits are within 1144 ASes because there are 421 ASes in the intersection of the guard and exit AS sets). There are $200 \times 962 = 192400$ virtual links from exit ASes to cluster representative of all Internet ASes. There are 4.053 AS or IXP organizations on each such virtual link on average. Assume again that the adversaries can be specified in two bytes. Then an explicit listing of the relevant adversaries on these links would be of size $192400 \times 4.053 \times 2 = 1.49\text{MiB}$. Of course, this could potentially change with every routing change, AS/IXP reorganization, and change in the Tor relay population. One would have to decide how often to update and how to do updates efficiently (for example, using virtual links diffs). The Tor directory system has evolved to optimize to account for information that needs to be updated relatively infrequently and to minimize the size of updates that are needed more often. We imagine a similar evolution of default trust information.

Note also that we have been very conservative in our estimation of needed download information. taking no advantage of redundancies or compression. Nonetheless, in addition to considering more than just path independence, our trust aware approach has efficiency advantages over prior AS-aware approaches because we only need to consider paths to and from client or destination cluster representatives. For example, even for the smaller Tor network of 2011, LASTor estimated an initial download requirement for clients of 13MB, with updates of 1.5MB when AS path length changed by about 5%, roughly once per week [4]. The download requirements for [3] were much smaller than LASTor's, but still several times our 1.5MiB. Other comparative observations will be made in Section IX.

IX. RELATED WORK

Distribution of trust across networks to secure the confidentiality of who is communicating with whom while tolerating compromise in parts of the network has its modern origins in Chaum's seminal work on mixes [34]. For most work in the last three and a half decades, trust is treated as uniformly distributed. Some mention the prospect of leveraging differences in trust. For example, when guard nodes were introduced into the Tor design, the advantages of choosing "entry guard nodes based on trust in the node administrator" is discussed [35], but there is no mathematical treatment of how trust is to be represented or reasoned about.

Such a treatment was introduced in 2009 when the concept of trust that we use in this paper was introduced, the complement to the a priori probability of compromise [7]. Johnson and Syverson provided a general analysis of end-to-end correlation when network nodes can have different degrees of trust and showed optimality results when there are just

two different levels of trust, depending on the trust levels and the relative size of the sets at each level. It also showed the unlikely prospect for a practical general analytic result for trust. Fuller adversary models and a more tractable routing algorithm based on the same model followed a few years later [8]. The expanded adversary model included the introduction of an adversary on the links between network nodes. Though trust in links was expressible, security analysis with respect to links was limited to cases assuming either that source links or destination links are observed.

Another approach to trust for anonymous communication is to explicitly leverage social network relations. Danezis *et al.* [36] described this for interactive but low-volume anonymous communication. Concentrating on low-volume applications allowed them to make use of padding, which is generally too expensive and too ineffective for some of the more popular application that use Tor. Mittal *et al.* [37] describe a social-network onion-routing architecture designed for Web browsing and other interactive communications that adds resistance to an active adversary. Both [37] and [8] use potentially much longer paths than Tor's three hops to achieve intended security, and performance may suffer significantly as a result of this and other features of the design.

Though [8] introduced for a trust-based approach the abstract representation of links to the network and adversary models, the structure of those links was unexplored. But the threat of link-level adversaries to Tor was recognized much earlier [2]. Feamster and Dingledine introduced the *location independence* metric for Tor connections, reflecting the probability that source and destination links will traverse the same AS. They looked at a handful of U.S. commercial ISPs for source links and a few dozen destinations they thought interesting or likely and found that a single AS would appear on both ends of a connection about 10% to 30% of the time. Edman and Syverson also considered location independence, but they also evaluated the most common source and destination ASes based on empirical measurements rather than simply intuitive plausibility [3]. They also evaluated if location diversity of Feamster and Dingledine's source and destination ASes was significantly affected by the growth of the network from about 30 to about 1300 relays. It wasn't, in fact for many source-destination pairs location diversity actually diminished. Finally, they introduced the first AS-aware routing algorithms. By using a partial snapshot of AS topology relationships and of IP prefix to AS mappings they were able to keep the amount of AS information clients needed to download for AS-aware route generation to about the same amount as the descriptor information typically downloaded. The routing algorithm discards entry-exit pairs that yield a common AS on both source and destination links for paths so inferred, which was shown to greatly reduce the probability that a single AS could observe both ends of a connection.

Akhoondi *et al.* propose a geographic-based relay selection method called LASTor [4] that ensures AS diversity in selected paths by relying on concise Internet atlases. LASTor is also intended to enhance performance by using geographic location to reduce path latencies. However, Wacek *et al.* showed that LASTor performance was actually not very good, even if routes ignored AS diversity, which should mean their performance evaluation of LASTor errs on the generous side [38].

Murdoch and Zieliński introduce consideration of IXPs on network links as another potential adversary. AS based location

diversity may be inadequate because a single IXP may observe traffic traversing multiple ASes [5]. They showed that an IXP positioned adversary can correlate traffic even at the low rates of sampling that the volumes passing through IXPs would seem to require. Link adversaries at both ASes and IXPs were extended by Johnson *et al.* to consider adversaries controlling multiple ASes or IXPs, such as companies that own many IXPs [11]. Juen [6] conducts a thorough study of AS and IXP adversaries similar to [11]. While [11] combines the evaluation with relay adversaries, including families of relays, Juen builds on the path prediction algorithm of [3] to more accurately predict the threat and to incorporate IXPs. He thus introduces the first Tor routing algorithms to resist both AS and IXP adversaries.

The prior work that has the most in common with this paper is [14]. That is the first work to connect the probabilistic trust of [8] with the detailed models of AS and IXP adversaries of [6] and [11]. In this paper we have focused on presenting a trust-aware path selection algorithm and security and performance analyses of that algorithm. Such are briefly mentioned by Jaggard *et al.*, but they are primarily focused on a formal language to express trust beliefs, an ontology of network elements to represent the threats to anonymity for Tor, and a technique to convert trust beliefs to a concise probabilistic representation of trust based on the ontology.

X. CONCLUSION

In this paper, we presented TAPS, a novel trust-aware path selection algorithm for Tor that enables clients to utilize the trust it has in network nodes and elements to significantly enhance their security. We analyze both the security and performance effects of multiple deployment scenarios, analyze how errors in trust affect the system, and explain both our trust model and how trust values could be obtained and propagated to users. Continued exploration of trust errors and their effects will inform the development of useful collections of trust beliefs, and experiments on the live network would provide us with more information about the usefulness of TAPS in practice.

ACKNOWLEDGMENTS

The work at the U.S. Naval Research Laboratory was supported by the Office of Naval Research. Joan Feigenbaum was supported by NSF grant CNS-1409599.

REFERENCES

- [1] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *IEEE Symposium on Security and Privacy (S&P 2005)*, May 2005.
- [2] N. Feamster and R. Dingledine, "Location diversity in anonymity networks," in *2004 ACM Workshop on Privacy in the Electronic Society (WPES 2004)*, October 2004.
- [3] M. Edman and P. Syverson, "AS-awareness in Tor path selection," in *16th ACM Conference on Computer and Communications Security (CCS 2009)*, November 2009.
- [4] M. Akhoondi, C. Yu, and H. V. Madhyastha, "LASTor: A low-latency AS-aware Tor client," in *IEEE Symposium on Security and Privacy (S&P 2012)*, May 2012.
- [5] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by Internet-exchange-level adversaries," in *7th Privacy Enhancing Technologies (PETs 2007)*, June 2007.
- [6] J. P. J. Juen, "Protecting anonymity in the presence of autonomous system and internet exchange level adversaries," Master's thesis, University of Illinois at Urbana-Champaign, 2012.
- [7] A. Johnson and P. Syverson, "More anonymous onion routing through trust," in *22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, July 2009.

- [8] A. Johnson, P. Syverson, R. Dingledine, and N. Mathewson, "Trust-based anonymous communication: Adversary models and routing algorithms," in *18th ACM Conference on Computer and Communications Security (CCS 2011)*, October 2011.
- [9] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on Tor using long paths," in *18th USENIX Security Symposium*, August 2009.
- [10] J. Geddes, R. Jansen, and N. Hopper, "How low can you go: Balancing performance with anonymity in Tor," in *13th Privacy Enhancing Technologies (PETS 2013)*, July 2013.
- [11] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on Tor by realistic adversaries," in *20th ACM conference on Computer and Communications Security (CCS 2013)*, November 2013.
- [12] R. Jansen and N. Hopper, "Shadow: Running Tor in a box for accurate and efficient experimentation," in *19th Annual Network and Distributed System Security Symposium (NDSS 2012)*, February 2012.
- [13] "Shadow Homepage," <http://shadow.github.io/>.
- [14] A. D. Jaggard, A. Johnson, S. Cortes, P. Syverson, and J. Feigenbaum, "20,000 in league under the sea: Anonymous communication, trust, MLATs, and undersea cables," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 1, April 2015.
- [15] "Tor directory protocol," <https://gitweb.torproject.org/torspec.git/blob/HEAD:/dir-spec.txt>.
- [16] X. Cai, J. Heidemann, B. Krishnamurthy, and W. Willinger, "An organization-level view of the Internet and its implications (extended)," USC/Information Sciences Institute, Tech. Rep. ISI-TR-2009-679, June 2012.
- [17] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *18th ACM Conference on Computer and Communications Security (CCS 2011)*, October 2011.
- [18] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES 2013)*, November 2013.
- [19] "Tor security advisory: 'relay early' traffic confirmation attack," <https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack>, July 2014.
- [20] M. Alsabah, K. Bauer, T. Elahi, and I. Goldberg, "The path less travelled: Overcoming Tor's bottlenecks with traffic splitting," in *13th Privacy Enhancing Technologies Symposium (PETS 2013)*, July 2013.
- [21] "Tor Metrics Portal," <http://metrics.torproject.org/>.
- [22] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport," in *23rd USENIX Security Symposium*, August 2014.
- [23] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for K-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, March 2009.
- [24] "Tor Project: manual," <https://www.torproject.org/docs/tor-manual.html.en>.
- [25] "The CAIDA UCSD Internet Topology Data Kit - December 2013," <http://www.caida.org/data/internet-topology-data-kit>.
- [26] "University of Oregon route views project," <http://www.routeviews.org/>.
- [27] J. Qiu and L. Gao, "AS path inference by exploiting known AS paths," in *IEEE GLOBECOM*, November 2005.
- [28] B. Augustin, B. Krishnamurthy, and W. Willinger, "IXPs: Mapped?" in *9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC 2009)*, November 2009.
- [29] "TorPerf," <https://gitweb.torproject.org/torperf.git/>.
- [30] "Shadow Git Repository," <https://github.com/shadow/shadow>.
- [31] "Shadow Tor plug-in Git Repository," <https://github.com/shadow/shadow-plugin-tor>.
- [32] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, January 2010.
- [33] G. Huston, M. Rossi, and G. Armitage, "Securing BGP — a literature survey," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 2, Second Quarter 2011.
- [34] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 4, no. 2, February 1981.
- [35] L. Øverlier and P. Syverson, "Locating hidden servers," in *IEEE Symposium on Security and Privacy (S&P 2006)*, May 2006.
- [36] G. Danezis, C. Diaz, C. Troncoso, and B. Laurie, "Drac: An architecture for anonymous low-volume communications," in *10th Privacy Enhancing Technologies Symposium (PETS 2010)*, July 2010.
- [37] P. Mittal, M. Wright, and N. Borisov, "Pisces: Anonymous communication using social networks," in *20th Annual Network & Distributed System Security Symposium (NDSS 2013)*, February 2013.
- [38] C. Wacek, H. Tan, K. Bauer, and M. Sherr, "An empirical evaluation of relay selection in Tor," in *20th Annual Network & Distributed System Security Symposium (NDSS 2013)*, February 2013.